uFLIP-OC: Understanding Flash I/O Patterns on Open-Channel Solid-State Drives

Ivan Luiz Picoli
IT University of Copenhagen
Denmark
ivpi@itu.dk

Carla Villegas Pasco
IT University of Copenhagen
Denmark
carv@itu.dk

Björn Þór Jónsson
IT University of Copenhagen
Denmark
bjth@itu.dk

Luc Bouganim
INRIA Saclay & UVSQ
France
Luc.Bouganim@inria.fr

Philippe Bonnet
IT University of Copenhagen
Denmark
phbo@itu.dk

ABSTRACT

Solid-State Drives (SSDs) have gained acceptance by providing the same block device abstraction as magnetic hard drives, at the cost of suboptimal resource utilisation and unpredictable performance. Recently, Open-Channel SSDs have emerged as a means to obtain predictably high performance, based on a clean break from the block device abstraction. Open-channel SSDs embed a minimal flash translation layer (FTL) and expose their internals to the host. The Linux open-channel SSD subsystem, LightNVM, lets kernel modules as well as user-space applications control data placement and I/O scheduling. This way, it is the host that is responsible for SSD management. But what kind of performance model should the host rely on to guide the way it manages data placement and I/O scheduling? For addressing this question we have defined uFLIP-OC, a benchmark designed to identify the I/O patterns that are best suited for a given open-channel SSD. Our experiments on a DragonFire Card (DFC) SSD, equipped with the OX controller, illustrate the performance impact of media characteristics and parallelism. We discuss how uFLIP-OC can be used to guide the design of host-based data systems on open-channel SSDs.

CCS CONCEPTS

• **General and reference** → **Evaluation**; • **Information systems** → **Information storage systems**; **Flash memory**;

KEYWORDS

Open-channel SSDs, NAND Flash, Benchmarking, uFLIP-OC

ACM Reference format:
Ivan Luiz Picoli, Carla Villegas Pasco, Björn Þór Jónsson, Luc Bouganim, and Philippe Bonnet. 2017. uFLIP-OC: Understanding Flash I/O Patterns on Open-Channel Solid-State Drives. In *Proceedings of APSys '17, Mumbai, India, September 2, 2017,* 8 pages.
https://doi.org/10.1145/3124680.3124741

APSys '17, September 2, 2017, Mumbai, India

ACM ISBN 978-1-4503-5197-3/17/09.
https://doi.org/10.1145/3124680.3124741

1 INTRODUCTION

Solid State Drives (SSDs) have replaced magnetic disks in data centers. Cloud providers now expect SSDs to provide predictably high performance, as well as high resource utilisation, for their dynamic workloads. As traditional SSDs offer the same interface as magnetic hard drives to abstract a radically different physical storage space, however, resource utilisation is suboptimal and performance is often unpredictable [6, 9]. An emerging option for fulfilling the requirements of cloud providers is based on open-channel SSDs, which expose their media geometry and parallelism to the host [1]. As it is the host's responsibility to manage data placement and I/O scheduling, it becomes possible to avoid redundancies and exploit optimisation opportunities in the storage stack. The question is then: how should a data system that relies on open-channel SSDs be designed? More precisely, the question is whether some I/O patterns should be favoured, while others should be avoided. This is the question studied in this paper.

Recently, He et al. [5] discussed the "unwritten contract" of traditional SSDs, i.e., SSDs equipped with an embedded Flash Translation Layer, that provide the block device abstraction (initially defined for magnetic hard drives): a linear space of logical block addresses (LBAs) associated with read and write operations. According to He et al., systems implemented on top of SSDs should follow five rules: (i) *request scale rule*: submit large requests or many outstanding requests, (ii) *locality rule*: favour locality to minimise misses in the FTL mapping table, (iii) *aligned sequentiality*: write sequentially within a block, (iv) *grouping by death time*: group on the same blocks data that is updated or deleted together, and (v) *uniform data lifetime*: favour data structures where data are updated/deleted in batch. But do these five rules still apply on open-channel SSDs? And if not, then what rules *do* apply?

Before we can answer these questions, however, we need a tool to understand the performance characteristics of open-channel SSDs. Bouganim et al. defined the uFLIP benchmark in 2009, as a means of characterizing the performance of flash-based SSDs [2]. More specifically, the goal was to understand the impact of the FTL on the performance of simple I/O patterns. As it turned out, the benchmark showed that different SSDs behaved in different ways and that the complexity of the FTL introduced significant performance variability. With open-channel SSDs, however, the FTL is out of the equation. Furthermore, while the simple I/O patterns defined

in the uFLIP micro-benchmarks could possibly yield a performance model, the uFLIP benchmark assumes a block device abstraction which is not supported by open-channel SSDs. Note that existing papers focusing on SSD performance and error patterns, such as Meza et al. [7], Ouyang et al. [8] or Grupp et al. [4] all make similar assumptions. In Linux, LightNVM instead introduces the PPA interface, a new interface that relies on a hierarchical address space and vector data commands (each read or write command can target up to 64 addresses).

In this paper, we redesign the uFLIP benchmark for the PPA interface. More specifically, we make the following contributions:

(1) We design uFLIP-OC, a variant of the uFLIP benchmark, adapted to the characteristics of the PPA interface of open-channel SSDs (Section 3).
(2) We apply the uFLIP-OC benchmark on an open-channel SSD composed of the DFC equipped with the OX controller (Sections 4 and 5).
(3) We revisit the five rules of He et al. and discuss the path towards a new performance contract for open-channel SSDs based on the uFLIP-OC benchmark (Section 6).

2 BACKGROUND

In this section we briefly review the main characteristics of open-channel SSDs and the uFLIP benchmark before proceeding to our contributions.

2.1 Open-Channel SSDs

Solid-State Drives (SSDs) are composed of tens of storage chips wired in parallel to a controller through a number of channels. Each storage chip can be abstracted as the minimal unit of parallelism in the SSD (also called LUNs). NAND flash LUNs are organized in planes, blocks and pages. Persistent memory, such as ST-MRAM or 3D Xpoint, is expected to be organised as a collection of sectors. Open-channel SSDs expose their internals to the host.

LightNVM, the Linux open-channel SSD subsystem, defines the Physical Page Address (PPA) interface that differs from the traditional block device abstraction in two ways. First, the address space is hierarchical. Each address specifies channel and LUN, as well as a media-specific address. For NAND-flash, each PPA contains channel, LUN, block, plane, page and sector. Open-channel SSDs can make the dimensions of their address space known (i.e., number of channels, number of LUNs per channel, number of planes per LUN, number of blocks per plane, number of pages per block, number of sectors per page). Second, the PPA interface supports vector I/Os. Read and write requests can be applied on up to 64 PPAs at a time. Note that write requests must be issued at page granularity, while the read request granularity can encompass any number of sectors.

2.2 uFLIP

uFLIP defines a collection of nine micro-benchmarks, each composed of a few I/O patterns. An I/O pattern is a sequence of I/Os, where each I/O is defined by: the *time* at which it is submitted (in this paper, we only consider consecutive patterns, where a thread submits a new I/O as soon as the previous one has completed); the *I/O size* (by default we consider 4KB I/Os); the *I/O mode* (read or write); and the *address* at which the I/O is targeted. Each micro-benchmark focuses on varying one of these I/O parameters, such as address alignment, locality, delays, order and parallelism. The performance of the specific I/O patterns then together define the performance characteristics of the SSD. In the original uFLIP benchmark the addresses are defined in the logical block address (LBA) space exposed by SSDs with embedded FTLs. In the next section, we revisit the uFLIP benchmark in the context of the PPA interface.

3 BENCHMARK DESIGN

The requirements of uFLIP-OC are derived from the characteristics of open-channel SSDs:

- With open-channel SSDs, media characteristics are exposed to the host. We should explore their impact on performance.
- The PPA interface supports vector I/Os. We should compare the parallelism obtained with vector I/Os to the parallelism obtained with a number of concurrent outstanding requests.
- I/Os are partitioned in the PPA space across channels and LUNs. We should explore the characteristics of intra-channel and inter-channel parallelism.

We define uFLIP-OC as a collection of four micro-benchmarks, organised in two thematic groups that focus on: (i) media characteristics and (ii) parallelism. Each micro-benchmark consists of a sequence of I/Os, at page granularity, on a given block. The blocks involved in a benchmark are erased prior to its execution. We do not consider random patterns, as they are not directly supported on open-channel SSDs. In the uFLIP terminology, we consider partitioned sequential reads or writes executed in parallel by varying number of threads. Table 1 summarizes the uFLIP-OC benchmark. Note that while the table defines a specific range of values that apply to the device under study in this paper, the micro-benchmarks can be adapted to any device geometry.

3.1 Media Characteristics

There is significant heterogeneity in the various non-volatile memories that compose the storage chips at the heart of open-channel SSDs. NAND-flash can be available as SLC (single bit per cell), MLC/TLC/QLC (two/three/four bits per cell) or 3D NAND (a three dimensional array of cells that each stores one or more bits). Different NAND-flash types exhibit different performance and endurance characteristics. Other types of non-volatile memories, such as ST-MRAM or 3D-Xpoint, only increase this heterogeneity. By design, these characteristics are exposed to the host.

Data sheets, if available, can help set out expectations for (i) performance, with a latency range for page writes/reads, as well as (ii) endurance, with guaranteed minimum number of program/erase cycles per block. But they cannot be used to characterise the interplay of reads and writes or to characterise the impact of wear on performance. We thus design two micro-benchmarks to identify these characteristics.

μOC_0: Read/Write Performance of a single LUN. This first micro-benchmark is executed by a single thread, accessing a single LUN. There is no form of parallelism. We focus on the latency and throughput of reads and writes at page granularity. We consider various mixes of reads and writes ranging from 100% read, to 100%

Table 1: uFLIP-OC micro-benchmark overview. Unspecified geometry components can be selected arbitrarily.

Name	Parallelism	Pattern	Threads	Pages	**Definition** (":" = loop, "()" = grouping, ";" = order, and "|" = choice)
μOC_0	No	Sequential	1	1	Block=0–63: Page=0–511: (WWWW | WWWR | WRWR | WRRR | RRRR)
μOC_1	No	Sequential	1	1	Loop: (Erase; Page=0–511: W; Page=0–511: R)
μOC_2	Multi-LUN	Round-Robin	1	1	Block=0–63: Page=0–511: LUN=0–3: (W | R)
	Threads	Round-Robin	1, 2, 4	1	Block=0–63: Page=0–511: LUN=T_i: (W | R) — T_i is the thread identifier
μOC_3	Vector I/Os	Round-Robin	1	1, 2, 4, 8	Block=0–63: Page=0–511: Channel=0–7: (W | R) — I/Os are issued as vectors
	Threads	Round-Robin	1, 2, 4, 8	1	Block=0–63: Page=0–511: Channel=T_i: (W | R) — T_i is the thread identifier

writes with three intermediate mixes of reads and writes (25% reads/75% writes, 50% reads/50% writes, 75% reads/25% writes). Our goal is twofold. We aim at characterizing (i) latency variance on a LUN with different mixes of read and write operations, and (ii) throughput variance across LUNs on the SSD.

μOC_1: Impact of Wear. This micro-benchmark sacrifices a block to study the impact of wear on performance. It focuses on a single block, accessed by a thread that loops through cycles of erase, writes and reads on the entire block, until an erase fails and the block is definitely classified as a bad block. Our goal is to trace the evolution of erase, write and read latency as a function of erase cycles, as well as the number of failures of page reads/writes.

3.2 Parallelism

Parallelism is the essence of SSDs: storage chips are wired in parallel onto each channel, several channels are wired in parallel to the controller, and the controller is multi-threaded. As we observed above, we should explore parallelism (i) within and across channels, as well as (ii) parallelism due to vector I/Os (a read/write command applied on multiple PPAs) and concurrent outstanding I/Os (either asynchronous I/Os or I/Os submitted by different threads). We design two micro-benchmarks to characterise the impact of parallelism on performance.

μOC_2: Intra-Channel Parallelism. This micro-benchmark focuses on parallelism across LUNs, within a channel. A single thread issues write I/Os at page granularity on a number of LUNs within a channel in round-robin fashion[1]. The number of LUNs targeted, as well as the modality of the I/Os (read or write) are the factors in this experiment. The measurements focus on throughput.

μOC_3: Inter-Channel Parallelism. This micro-benchmark focuses on parallelism across channels. A number of threads issue I/Os at page granularity on a single LUN per channel. There are three factors in this experiment: the number of submitting threads (from one to the number of channels), the number of PPAs targeted in each I/O (ranging from the number of PPA per page to 64 PPA addresses), and the modality of each I/O (read or write).

4 EXPERIMENTAL FRAMEWORK

We apply the uFLIP-OC benchmark to an open-channel SSD, based on the DFC equipped with the OX controller. This is the first non-commercial open-channel SSD available to the research community. Other open-channel SSDs include research prototypes (Cosmos OpenSSD[2]), as well as commercial systems (CNEX Labs Westlake, Radian RMS-325). Evaluating these systems with uFLIP-OC is a topic for future work. In this section, we give a brief presentation of the DFC and the OX controller. In the next section, we present the results of the benchmark.

4.1 DragonFire Card (DFC)

The DragonFire Card (DFC) is a programmable SSD device, designed by DellEMC and NXP. It is composed of two boards: a main board and a storage board. The main board is equipped with an LS2088A SoC, based on ARMV8, 16 GB of RAM and an SD card. It provides connectivity via 4 PCIe Gen3 lanes and 4x10G Ethernet. A Linux variant, with a block device controller and full-fledged FTL, runs on the main board. The storage board is composed of an FPGA board, equipped with an embedded storage controller for 4 DIMM slots. The storage board is connected to the main board via 2x4 PCIe Gen3 lanes. The storage board can be equipped with various forms of DIMMs: RAM, Flash or other forms of NVM. For the experimental results, we have equipped the storage board with two modules of four Micron MLC NAND chips each, organized in 16 KB pages, 512 pages per block, 2048 blocks per LUN spread in 2 planes, and 4 LUNs per chip. The total storage capacity for all eight channels is 512 GB.

An open-source community[3] is organized around this hardware platform with teams working on a wide range of issues, from open-channel SSDs to video processing and genomics. For example, VVDN provides support for the Linux installation.

4.2 OX Controller

OX is the first open-source LightNVM-enabled NVMe controller. It has been designed to execute I/O commands in parallel. Figure 1 shows the I/O command flow within the controller; colors represent thread responsibilities.

- BLUE threads consume the NVMe queue located on the host. After dequeuing the NVMe command from the host memory queue, the blue thread dispatches it to an OX submission thread (red). OX has one BLUE thread per host core.
- RED threads translate NVMe commands into commands that are submitted to the media layer (purple). On DFC, the media layer is located inside the storage card FPGA, which is accessed via an interface library. For our experiments, we only

[1] Thus eliminating concurrency issues that we consider in μOC_3

[2] http://www.openssd-project.org/wiki/Cosmos_OpenSSD_Platform

[3] https://github.com/DFC-OpenSource

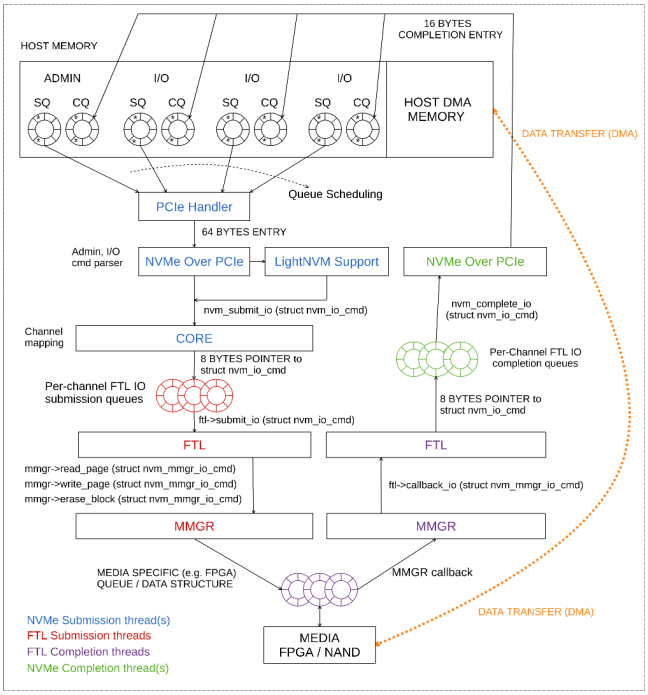

Figure 1: I/O Flow within the OX controller

consider NVMe commands that conform to the LightNVM specification. OX has one RED thread per channel.

- PURPLE threads are responsible for media I/O completion. These threads check for completed NAND I/O commands in the purple queues and post completion to the appropriate completion queue (green). Note that for data commands, data is transferred directly between host memory and the FPGA via DMA.
- GREEN threads are responsible for NVMe I/O completion. These threads check for completed I/O commands in the green queues and post a completion NVMe queue entry in the host memory.

We consider one BLUE thread per host core, and one thread per channel for RED, PURPLE and GREEN threads. OX is released as an open-source project.[4]

4.3 Workload

We have defined a tool, called FOX, to submit the uFLIP-OC I/O patterns on open-channel SSDs. FOX is a user-space tool that relies on the liblightnvm library[5] to submit I/Os to open-channel SSDs via LightNVM. Liblightnvm relies on IOCTL calls for submitting commands. As a result, each I/O is synchronous. We thus rely on commands submitted by multiple threads to obtain concurrent outstanding I/Os. FOX is open-source and available to the community[6].

[4]https://github.com/DFC-OpenSource/ox-ctrl
[5]https://github.com/OpenChannelSSD/liblightnvm
[6]https://github.com/DFC-OpenSource/fox

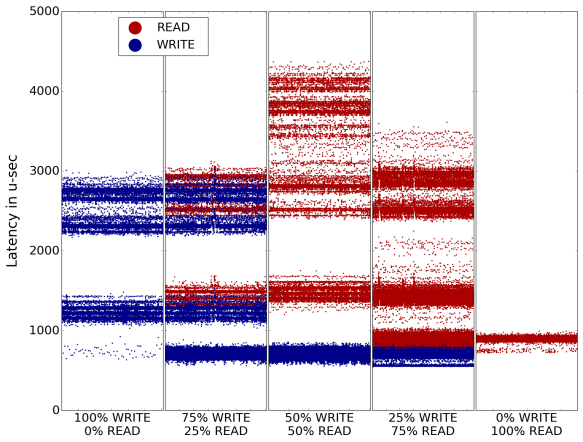

Figure 2: μOC_0: Impact of read/write mix on latency.

5 EXPERIMENTAL RESULTS

In this Section, we present the results of the uFLIP-OC benchmark applied to the DFC equipped with OX. We start by discussing the impact of media characteristics and then the impact of parallelism. We discuss lessons we can derive from these results about the SSD performance contract of He et al. [5] in the next section.

5.1 Media Characteristics

5.1.1 Latency Variance. We apply μOC_0 and measure latency for various mixes of read and write operations. Recall that each I/O is executed at page granularity. A mix of 25%R and 75%W corresponds to a sequence of three writes followed by one read. In this micro-benchmark, a single thread submits I/Os to a single LUN and a given channel. Based on the data sheets of the NAND chips, we expect that writes take between 1.6 and 3.0 msec while reads take approximately 150 usec. The write characteristics are due to the nature of the MLC NAND chip, which stores two bits per cell, and the first ("low") bit must be written before the second ("high") bit. As the MLC chip exposes pairs of pages encoded on the *same cells*, the consequence is that (i) pairs of pages must be written together and (ii) the "low page" is written before the "high page". So, we expect that write latency will oscillate between two values and that read latency will be stable and low. Existing work on open-channel SSDs [1] suggests that reads will be slowed down by writes. Figure 2 presents the results of μOC_0 for the various mixes of read and write operations. In each experiment, the number of I/Os submitted is equal to 32768, which corresponds to writing or reading 64 blocks. For 100% reads, we observe stable latency but much higher than what could be expected from the data sheet. This suggests that the overhead associated with reads (essentially ECC check) is significant. For 100% writes, we observe three bands: (i) from 2.2 to 3.0 msec, (ii) from 1.0 to 1.3 msec and (iii) around 800 usec. The first two bands correspond to what we expect for high and low pages. The third band corresponds to write latency below what is expected from the NAND chip. Our hypothesis is that some form of write-back is implemented within the DFC. Write performance for mixes of reads and writes confirm this hypothesis. Any mix of read and writes reinforces this third band, which is

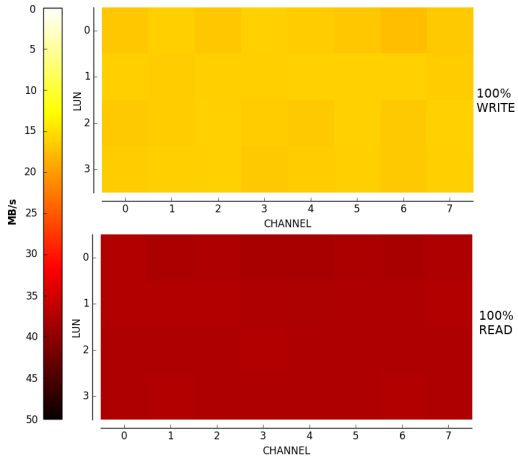

Figure 3: μOC_0: Heatmap of throughput for the entire SSD.

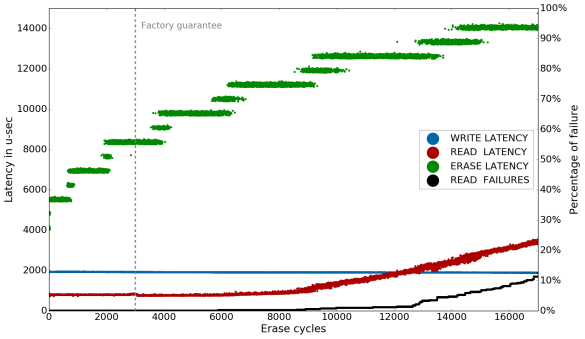

Figure 4: μOC_1: Impact of wear (erase cycles) on latency (left axis) and read failures (right axis).

faster than NAND. As soon as the ratio of reads is greater than 50%, then the latency of writes is stable below 1.0 msec while read latency increases dramatically. Reads are blocked by writes and the cost of NAND writes is reflected in the latency of reads. At 50% reads, read latency can reach above 4 msec. Such degradation in performance does not result from the hidden cost of writes alone. We observe here the result of read disturbances, where reads must be retried because of interference from writes.

5.1.2 Throughput Variance. We apply μOC_0 on every LUN for all channels and measure throughput. We focus on 100% read and 100% write workloads. Our goal is to visualise variance across LUNs in the SSD. Figure 3 shows a heatmap to represent the result. We feared that performance would be uneven because we tend to experiment mostly with channel 0 and LUN 0 on each channel. But the results show little variance across LUNs. Throughput is stable at 16 MB/sec per LUN for writes and 38 MB/sec per LUN on reads. Note that this throughput is the result of sequential, synchronous I/Os on one LUN at a time. There is no form of parallelism involved.

5.1.3 Wear. In order to measure the impact of wear (i.e., the number of erases performed on a block) on performance, we sacrifice a block and conduct μOC_1. While we do not really know the state of the block we choose for this experiment, it is one of the less used blocks of the system. The NAND flash data sheet indicates a guarantee of 3,000 erase cycles per block. Based on Cai's outstanding study of NAND flash errors[3], we expect that the open-channel SSD will exhibit a low number of failures up to a point where the number of failures will increase steeply and negatively impact the performance of all operations.

Figure 4 shows the result of μOC_1. We observe the first read failure only after 5,872 erases, or almost double the factory guarantee of the underlying NAND. This shows that ECC introduces high latency for reads but provides perfect error correction until wear reaches a given threshold. We remark that this tradeoff between read latency (due to ECC) and read failure rate is a key characteristic of open-channel SSDs. On the other hand, the erase process must apply increasingly larger voltages to avoid failure. The voltages are applied in a stepwise fashion, so the cost of erase operations increases regularly throughout the experiment. So, on the DFC equipped with the current generation of NAND chips, there is a correlation between erase latency and failure rate. This result suggests that it might possible to assume that reads never fail until erase latency reaches a given threshold. This would have a major impact on the design of host-based FTLs or application-specific FTLs that today assume that I/Os might fail at page, block or die level and deploy considerable engineering resources to design failure handling mechanisms.

A surprising outcome of this experiment is that write latency remains constant and unaffected by wear. This is a worrying characteristic that can be linked to the write-back mechanism identified with μOC_0. Writes always complete fast, and failures are only identified on reads. Note, however, that this behaviour makes sense under the assumption that reads never fail.

Finally, note that we stopped the experiment after 17,000 erase cycles; at this point the failure rate for reads had reached 10%.

5.2 Parallelism

5.2.1 Intra-Channel Parallelism. We first focus on parallelism within a channel with μOC_2. Write or read I/Os at page granularity are sent to a varying number of LUNs within one channel in a round-robin manner, either using one thread or multiple threads. We expect that requests are executed in parallel on the different LUNs and that throughput increases in proportion to the number of LUNs until the channel becomes a bottleneck (i.e., writes are blocked until the channel is ready).

Table 2 shows the throughput of μOC_2 when targeting 1, 2 and 4 LUNs within a channel. Consider first write requests issued by multiple threads. Performance for 1 LUN is the same as in μOC_0 at approximately 16 MB/sec. While throughput increases nearly linearly with the number of LUNs, it does not increase by a factor corresponding to the number of LUNs. We believe that this must be due to overhead on the DFC and OX controller. With only a single thread issuing synchronous writes, throughput is nearly the same, due to write-back on the DFC; control is given back to the thread very quickly, but when the threads returns to LUN 0, it must wait for the completion of all previous writes.

Table 2: μOC_2: Impact of intra-channel parallelism on throughput.

	100% Writes Multiple Threads		100% Writes Single Thread		100% Reads Multiple Threads	
LUNs	Throughp. (MB/s)	Scaling Factor	Throughp. (MB/s)	Scaling Factor	Throughp. (MB/s)	Scaling Factor
1	16.81	—	16.81	—	37.09	—
2	27.22	1.62	23.47	1.40	49.77	1.34
4	37.85	2.25	33.45	1.99	49.73	1.34

Table 3: μOC_3: Impact of inter-channel parallelism on write throughput: Vector I/Os vs Multiple threads.

	Vectored I/O (Single Thread)			Parallel I/O (Multiple Threads)		
Total I/O size	Pages	Throughput (MB/s)	Scaling Factor	Threads	Throughput (MB/s)	Scaling Factor
32 KB	1	44.37	—	1	45.86	—
64 KB	2	80.35	1.81	2	94.37	2.06
128 KB	4	117.84	2.66	4	119.90	2.61
256 KB	8	128.59	2.90	8	127.23	2.77

For reads, the story is different, as synchronous reads must be completed before handing back control. With one thread (not shown) the throughput is not affected by the number of LUNs considered. With multiple thread, throughput is increased when two threads issue read requests in parallel; as the maximal throughput per channel is 50MB/s, further threads do not increase throughput.

5.2.2 Inter-Channel Parallelism. We now turn to parallelism across channels with μOC_3. We first explore the impact of vector I/Os (a single command applied to up to 64 PPAs) and compare it to the impact of outstanding concurrent I/Os submitted by different threads. On the DFC, equipped with MLC NAND, page granularity corresponds to 8 PPAs (1 PPA per sector, 4 sectors per page and 2 pages per plane). We thus experiment with vector I/Os applied to multiples of 8 PPAs (8, 16, 32 and 64). Each group of 8 PPAs corresponds to a page located on a separate channel, so our experiment targets 1, 2, 4 and 8 channels. We consider outstanding concurrent I/Os submitted by a thread dedicated to a given channel. We experiment with 1, 2, 4 and 8 threads so that potential inter-channel parallelism is the same for vector I/Os and concurrent I/Os. When the number of targeted channels is less than 8, the experiment targets each channel in turn in round-robin fashion.

Table 3 shows the write throughput for vectored and concurrent I/Os. First, we observe that even when a single channel is targeted at a time (8 PPAs or a single thread), throughput is more than 40 MB/sec, i.e., better than the throughput obtained with intra-channel parallelism. This is because targeting each channel in a round-robin fashion effectively hides a significant portion of the time spent writing on NAND. Less time is spent waiting for a LUN to become available and as a result throughput is increased. As expected, both vector I/O and concurrent I/O take advantage of inter-channel parallelism. The throughput when targeting two channels, with 16 PPAs or two threads, is twice the throughput obtained with 1 channel, with 8 PPAs or 1 thread. When targeting four or eight channels throughput is increased up to 130 MB/sec, but not by a factor of two when doubling the number of channels targeted. The throughput obtained with vector I/O and concurrent I/Os is similar. With 32 threads, each targeting a LUN (there are 8 channels and 4 LUNs/channel on the DFC), we reach 300 MB/sec throughput for writes and 400 MB/sec throughput for read. So, reaching maximum throughput for the device requires some level of concurrent I/Os, either due to asynchronous I/Os issued from the kernel (e.g. pblk) or multiple threads in user space via liblightnvm.

Figure 5 shows the latency obtained for various mixes of reads and writes issued with concurrent I/Os. More specifically, each thread issues either read or write on a separate LUN. We observe that read latency remains low, stable and unaffected by writes. As suggested by previous work [1] separating reads and writes leads to minimal latency variance. An interesting effect is observed, however, with 100%W, where the write times have a much less predictable latency than when mixed with reads, while throughput is not affected.

6 IMPACT ON PERFORMANCE CONTRACT

The unwritten SSD contract and the five rules identified by Hen et al. [5] were defined for SSDs equipped with embedded FTL. Let us revisit how these five rules apply to open-channel SSDs in light of the results of the uFLIP-OC benchmark applied on the DFC equipped with the OX controller.

(1) *Request scale rule:* Our results show that there is a tension between max throughput (that requires a queue of outstanding requests an each LUN) and low latency variance (that requires separation of writes from reads). The request scale rule does not allow to cope with this trade-off. It is up to system designers to consider data placement and I/O scheduling strategies that strike an appropriate balance for generic or application-specific FTLs.
(2) *Locality rule:* Locality might lead to interferences between reads and writes on a same LUN and thus high latency variance. This rule is thus not applicable.
(3) *Aligned sequentiality rule:* This rule still holds, and is indeed trivial to enforce with the PPA address space. Alignment within a block requires that writes start at page 0 in a given block.
(4) *Grouping by death time rule & Uniform data lifetime rule:* These rules focus on requirements for data placement; open-channel SSDs make it possible for the host to take such decisions without impediment. Our benchmark results, however, show another requirement on data placement: reads and writes should be isolated to preserve low latency variability.

Note that none of these rules account for media characteristics. Our results indicate that aggressive assumptions can be made about the absence of read failures in the upper layers of the system. More work is needed to generalise the results and identify whether a set of design rules, favouring specific I/O patterns, can be derived for open-channel SSDs in general. In particular, an open question is how different types of media, different generations of storage chips,

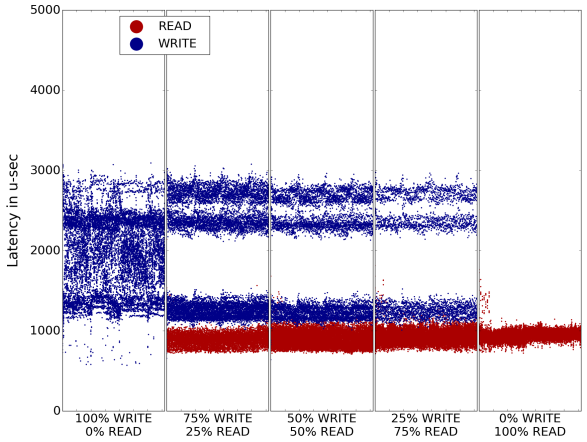

Figure 5: μOC_3: Impact of parallelism on latency for mixes of reads and writes.

or even different ECC design decisions will impact the performance of an open-channel SSD.

7 CONCLUSION

In this paper, we have presented the uFLIP-OC benchmark, a collection of micro-benchmarks designed to characterise the performance of open-channel SSDs. We have applied it to the DFC, the first publicly available non-commercial open-channel SSD, equipped with the OX controller, and discussed the results in detail. We believe that our micro-benchmarks can be used to define a new performance contract for open-channel SSDs. Our benchmark will also be useful for the designers of open-channel SSDs, or for customers comparing various open-channel SSDs for a given system. Future work includes a detailed study of the role of media characteristics on data systems design.

ACKNOWLEDGEMENT

This research was partially supported by the Coordination for the Improvement of Higher Education Personnel (CAPES), Brazil, who provided Ph.D fellowship for the first author.

REFERENCES

[1] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. 2017. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proceedings of the USENIX Conference on File and Storage Technologies, (FAST), Santa Clara, CA, USA*. 359–374.

[2] Luc Bouganim, B. Jónsson, and P. Bonnet. 2009. uFLIP: Understanding flash IO patterns. In *Proceedings of the Biennial Conference on Innovative Data Systems (CIDR)*.

[3] Yu Cai, Erich F Haratsch, Onur Mutlu, and Ken Mai. 2012. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 521–526.

[4] Laura M. Grupp, John D. Davis, and Steven Swanson. 2012. The Bleak Future of NAND Flash Memory. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, Berkeley, CA, USA, 2–2.

[5] Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2017. The Unwritten Contract of Solid State Drives. In *Proceedings of the European Conference on Computer Systems (EuroSys)*. ACM, New York, NY, USA, 127–144.

[6] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs Through OPS Isolation. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*. 183–189.

[7] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. 2015. A large-scale study of flash memory failures in the field. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 43. ACM, 177–190.

[8] Jian Ouyang, Shiding Lin, S Jiang, and Z Hou. 2014. SDF: Software-defined flash for web-scale internet storage systems. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*.

[9] Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman. 2014. Don't stack your log on my log. In *Proceedings of the Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW)*.